Si je connais “très” bien ? GeoGebra, je ne suis pas nécessairement un cador en JavaScript.
S’il s’agit d’aider un utilisateur, j’essaye de respecter au mieux son style.
Et ça me gonfle un peu de travailler dans le Javascript global, toujours sauvegarder, fermer et ouvrir à nouveau le fichier.ggb pour vérifier le bon déroulement du script.
L’éditeur n’est pas sympa, identation, identification des blocs, bon, paraît qu’en regardant le script dans notepad++ c’est plus clair ? Mais ça fait encore de la manipulation en plus !

Analyse du script créé au départ par JH pour sa princesse, puis modifié par moi.

```
function ggbOnInit() {
   ggbApplet.debug("ggbOnInit");
   ggbApplet.registerAddListener("NouvObj");
}
function NouvObj(obj) {
```

Ca commence comme ça …
On ajoute un mouchard de la création d’objet.
Dès qu’un objet est créé, il fait appel à la fonction, ici dénommée NouvObj.

```
   type = ggbApplet.getObjectType(obj);
   if (type == 'point') {
   cmd = ggbApplet.getCommandString(obj);
```

Ici, je teste si le nouvel objet créé est un point.

Si oui, je lis la commande qui le crée.

```
      if (cmd.indexOf('Intersection') != -1) {
         virg1 = cmd.indexOf(',');
         if (cmd.indexOf(',', virg1 + 1) != -1) {
            virg2 = cmd.indexOf(',', virg1 + 1);
         } else {
            virg2 = cmd.indexOf(')', virg1 + 1);
         };
         obj1 = cmd.substring(13, virg1);
         ggbApplet.setVisible(obj1, false);
         obj2 = cmd.substring(virg1 + 2, virg2);
         ggbApplet.setVisible(obj2, false);
```

Le but est de savoir si le point a été créé par intersection, indexOf('Intersection') retourne -1 si la chaîne Intersection n’est pas dans la commande.
Mais il y a 2 syntaxes Intersection(obj1, obj2) ou Intersection(obj1, obj2, n°).
Je demande la position virg1 de la 1ère virgule, puis celle de la 2ème, si elle existe, sinon de la) finale. (Les positions commencent à 0. Il y a toujours une espace après la virgule dans une commande.). Ce qui suit Intersection(commence à la position 13.
Je récupère donc, par substring, 2 sous-chaînes, noms des objets utilisés pour l’intersection, et je demande que ceux-ci ne soient pas affichés **setVisible** , pour nettoyer la figure.

```
         test1 = ggbApplet.getValue("test1");
.../...
à recopier pour 3, 5, 7, 9, 10, 12
.../...
         test13 = ggbApplet.getValue("test13");
         cmd1 = "SetValue[test1," + obj + "==
cible1]";
.../...
à recopier pour 3, 5, 7, 9, 10, 12
.../...
         cmd13 = "SetValue[test13," + obj + "==
cible13]";
```

Lecture des valeurs des objets **getValue** test_ du fichier.
Écriture des commandes, **SetValue** qui changeront leur valeur de 0 à 1 si la cible est atteinte, ce qui est testé par obj == cible.
(Un test d’égalité est créé par 2 signes “égal”.)

La commande cmd1 fait passer test1 de 0 à 1 si la cible1 est atteinte.

if (test1 == 0) {
ggbApplet.evalCommand(cmd1);
};
.../...
à recopier pour 3, 5, 7, 9, 10, 12
.../...
if (test13 == 0) {
ggbApplet.evalCommand(cmd13);
};
};
};
	Accolade de fermeture de if (test13 == 0) {
	Accolade de fermeture de if (cmd.indexOf
	(elle aurait pu, dû se trouver plus haut ?)
	Accolade de fermeture de if (type == 'point') {

if (type == 'triangle' || type == 'quadrilateral') {
test2 = ggbApplet.getValue("test2");
.../...
à recopier pour 4, 6, 8, 11
.../...
test14 = ggbApplet.getValue("test14");
cmd2 = "SetValue[test2 ," + obj + "== cible2]";
.../...
à recopier pour 4, 6, 8, 11
.../...
cmd14 = "SetValue[test14 ," + obj + "== cible14]";
if (test2 == 0) {
ggbApplet.evalCommand(cmd2);
};
.../...
à recopier pour 4, 6, 8, 11
.../...
if (test14 == 0) {
ggbApplet.evalCommand(cmd14);
};
};
ggbApplet.refreshViews();
}
	Ici, je teste si le nouvel objet créé est un triangle ou un quadrilatère.
	
	Comme pour les points :
	Lecture des valeurs des objets **getValue** test_ du fichier.
	Écriture des commandes, **SetValue** qui changeront leur valeur de 0 à 1 si la cible est atteinte, ce qui est testé par obj == cible.
	(Un test d'égalité est créé par 2 signes "égal".)
	
	La commande cmd2 fait passer test1 de 0 à 1 si la cible2 est atteinte.
	
	La syntaxe du if :
	
	If (condition) {
	instructions si vraie;
	} else {
	instructions si fausse;;
	};
	
	Cette une bonne habitude de faire rafraichir/réactualiser la construction.

Analyse du script créé au départ par JH pour son astre, puis modifié par moi.	

function ggbOnInit() {
ggbApplet.debug("ggbOnInit");
ggbApplet.registerAddListener("NouvObj");
}
function NouvObj(obj) {	

type = ggbApplet.getObjectType(obj); noel = ggbApplet.getCommandString(obj);	Je lis le type du nouvel objet et la commande qui le crée.

if (type == 'point') {
test1 = ggbApplet.getValue("test1");
.../...
à recopier pour 2, 3, 4
.../...
test5 = ggbApplet.getValue("test5");
cmd1 = "SetValue[test1," + obj + "==
cible1]";
.../...
à recopier pour 2, 3, 4
.../...
cmd5 = "SetValue[test5 ," + obj + "==
cible5]";
 	
if (test1 == 0) {
ggbApplet.evalCommand(cmd1);
};
.../...
à recopier pour 2, 3, 4
.../...
if (test5 == 0) {
ggbApplet.evalCommand(cmd5);
};
if (obj == 'D') {
virg1 = noel.indexOf(',');
virg2 = noel.indexOf(')', virg1 + 1);
obj1 = noel.substring(13, virg1);
ggbApplet.setVisible(obj1, false);
obj2 = noel.substring(virg1 + 2, virg2);
ggbApplet.setVisible(obj2, false);
}
};
	Si le nouvel objet est un point, je demande que la valeur d'un test de 1 à 5 soit mise à 1 (= vrai) si la cible point du fichier est atteinte.
	
	Dans le cas du point D, construit par intersection des 2 perpendiculaires, je lis le nom de ses 2 antécédents dans sa chaîne de construction.
	Et je leur transmets la propriété "Non affiché".

if (type == 'segment') { ord6 = "Segment(A, cible4)"; ord7 = "Segment(B, cible4)"; ord8 = "Segment(cible1, cible2)"; ord9 = "Segment(cible2, cible4)"; ord10 = "Segment(C, cible2)"; ord11 = "Segment(cible3, cible5)"; ord12 = "Segment(A, C)"; ord13 = "Segment(B, cible1)"; ord6r = "Segment(cible4, A)"; ord7r = "Segment(cible4, B)"; ord8r = "Segment(cible2, cible1)"; ord9r = "Segment(cible4, cible2)"; ord10r = "Segment(cible2, C)";	Si le nouvel objet est un segment, je vise les cibles 6 à 13 ! Le problème est qu'un test d'égalité de segments dans GeoGebra est effectué sur les longueurs !!! Je dois passer par les commandes, et en plus pour GeoGebra Segment(A, B) et Segment(B, A) sont des commandes différentes. Je dois donc définir tous les segments cible dans les 2 sens !!! (ord6 à 13, avec les retournées ord6r à 13r)

ord11r = "Segment(cible5, cible3)";
ord12r = "Segment(C, A)";
ord13r = "Segment(cible1, B)";
if (ord6 == noel || ord6r == noel) {
ggbApplet.evalCommand("test6=1");
};
.../...
à recopier de 7 à 12
.../...
if (ord13 == noel || ord13r == noel) {
ggbApplet.evalCommand("test13=1");
};
};
ggbApplet.refreshViews();
}
	Le test6 passera à 1 (=vrai) si la commande créant le nouveau segment correspond à l'une ou l'autre des commandes ord6 ou || ord6r

Analyse d'un petit script créé par moi, pour tester position d'un point.	

function ggbOnInit() {
ggbApplet.debug("ggbOnInit");
ggbApplet.registerAddListener("NouvObj");
}
function NouvObj(obj) {	

type = ggbApplet.getObjectType(obj);
 	
if (type == 'point') {
lex = Number(ggbApplet.**getXcoord**(obj));
ley = Number(ggbApplet.**getYcoord**(obj));
if (lex > 0 && ley > 0) {
ggbApplet.alert("Point dans le 1er quadrant");
}
if (lex < 0 && ley > 0) {
ggbApplet.alert("Point dans le 2ème quadrant");
}
if (lex < 0 && ley < 0) {
ggbApplet.alert("Point dans le 3ème quadrant");
}
if (lex > 0 && ley < 0) {
ggbApplet.alert("Point dans le 4ème quadrant");
}
if (lex == 0 && ley != 0) {
ggbApplet.alert("Point sur l'axe des ordonnées");
}
	Je lis le type du nouvel objet.
	
	Si le nouvel objet est un point, j'en lis ses coordonnées : lex et ley.
	Mais **getXcoord** et **getXcoord** retournent des chaînes de caractères, il faut donc coupler avec une commande JavaScript pour traiter en tant que nombre : Number.
	En JS && correspond au ET logique. alert provoque l'affichage d'un message.

`if (lex == 0 && ley == 0) {`
`  ggbApplet.alert("Point à l'origine");`
`}`
`if (lex != 0 && ley == 0) {`
`  ggbApplet.alert("Point sur l'axe des abscisses");`
`}`
`};`	

`if (type == 'segment') {`
`noel = ggbApplet.getCommandString(obj);`
`virg1 = noel.indexOf(',');`
`virg2 = noel.indexOf(')', virg1 + 1);`
`obj1 = noel.substring(8, virg1);`
`lex1 = Number(ggbApplet.getXcoord(obj1));`
`ley1 = Number(ggbApplet.getYcoord(obj1));`
`obj2 = noel.substring(virg1 + 2, virg2);`
`lex2 = Number(ggbApplet.getXcoord(obj2));`
`ley2 = Number(ggbApplet.getYcoord(obj2));`
`if (ley1 == ley2) {`
`  ggbApplet.alert("Segment parallèle à l'axe des abscisses");`
`} else {`
`  if (lex1 == lex2) {`
`  ggbApplet.alert("Segment parallèle à l'axe des ordonnées");`
`  };`
`};`
`};`
`ggbApplet.refreshViews();`
`}`
	Si le nouvel objet est un segment, j'en lis ses 2 antécédents dans la chaîne de commande.
	
	Puis après avoir déterminé les coordonnées de ces 2 points, je les compare afin de déterminer un éventuel parallélisme du segment construit avec un des axes.

Ce document n'est pas un brouillon, mais n'a pas non plus la prétention d'être d'une réalisation parfaite, même pendant le confinement mes journées n'ont que 24 heures. *Portez-vous bien !!!* *Noël ne sera pas normal cette année ! L'ai-je déjà été ?*